Exposé: Eigene Programmiersprache? WebAssembly macht's möglich!

Viele Entwickler träumen davon, ihre eigene Programmiersprache zu erschaffen, die perfekt an die eigenen Präferenzen und Bedürfnisse angepasst ist. Um dies umzusetzen, muss ein eigener Compiler programmiert werden, der den Quellcode in eine für Computer verständliche Darstellung umwandelt.
Die tragende Komponente dabei ist das Backend, welches dafür zuständig ist, die Anweisungen in Binärcode für die jeweilige Prozessorarchitektur umzuwandeln. Zum einen ist der Binärcode sehr schwer korrekt zu generieren und zum anderen ist er immer spezifisch für Betriebssystem und Prozessorarchitektur. So läuft ein Programm, das für einen Intel-x86_64-Prozessor unter Windows kompiliert ist, nicht unter MacOS auf einem M1-Arm-Prozessor.

Die im Jahr 2017 veröffentlichte Technologie WebAssembly (kurz WASM) versucht diese Probleme zu lösen. WASM bietet mit einem universellen Ausführungsformat ein plattformunabhängiges Ziel für Compiler. Wie der Name verrät, ermöglicht WebAssembly die Nutzung von Programmiersprachen neben Javascript im Browser.
Zusätzlich beinhaltet der WASM-Standard diverse Sicherheitsfeatures und Kompatibilität mit anderen Programmiersprachen wie Rust, C und C++, die schon zu WASM kompiliert werden können.

Jetzt stellt sich die Frage: “Senkt WASM die Einstiegshürden so stark, dass auch Amateurentwickler eigene Compiler realisieren können?”

Durch meine bisherigen Recherchen zu kompilierten Programmiersprachen wurde klar, dass die meisten Compiler von ganzen Teams oder Unternehmen entwickelt werden, während es eher ungewöhnlich ist, dass ein Hobby-Entwickler seine eigene Programmiersprache entwickelt.

Somit lautet meine Leitfrage: “**Inwiefern erleichtert WebAssembly den Bau eigener Compiler für Amateurentwickler?”**

In meiner Facharbeit will ich in einem Selbstversuch eine eigene simple Programmiersprache entwickeln, die den Code nach einer selbst definierten Syntax in Tokens unterteilt, diese in einem Baum ins Verhältnis zueinander setzt und letztendlich WASM-Bytecode generiert. Dieser WASM-Bytecode kann dann von einer existierenden Runtime, beispielsweise einem Webbrowser, ausgeführt werden. Für mein Projekt fokussiere ich mich bewusst auf einen kleineren Funktionsumfang und werde den WASM-Standard nicht voll ausreizen, um das Projekt in einem umsetzbaren Rahmen zu halten. So werde ich mich auf Arithmetik mit Ganzzahlen, Variablen und einfache Kontrollstrukturen beschränken.

In meiner Facharbeit werde ich zunächst in die grundlegenden Konzepte des Compilerbaus einführen und erklären, was einen Compiler ausmacht.
Dann gebe ich einen schnellen Überblick über WebAssembly und zeige, welche besonderen Chancen und Probleme sich daraus für den Compilerbau ergeben.
Im praktischen Teil entwickle ich einen einfachen Mini-Compiler, dessen Syntax und Funktionsumfang ich kurz definiere und anschließend in einem Selbstversuch umsetze.
Abschließend werde ich die zentralen Ergebnisse zusammenfassen und einen kurzen Ausblick auf die zukünftigen Möglichkeiten von Hobby-Programmiersprachen geben.

Folgender Zeitplan soll mir dabei als Orientierung dienen:

Zeitraum	**Arbeitsschritte**
Zuvor	genaue Definition von Syntax und Funktionsumfang
08.01.2026 – 14.01.2026	Umsetzung des Selbstversuchs
15.01.2026 – 21.01.2026	Debugging und Fertigstellung des Mini-Compilers
22.01.2026 – 28.01.2026	Formulierung erster Rohtext
29.01.2026 – 04.02.2026	Formulierung der Facharbeit
05.02.2026 – 11.02.2026	Formulierung der Facharbeit
12.02.2026 – 18.02.2026	Feinschliff der Facharbeit
19.02.2026	**Abgabe** der Facharbeit